Session Code: MLB318

mobile PC & devices

CLR Implementation Overview: Implementing CLR For Smart Personal Objects Technology (SPOT)

Donald Thompson
Architect / Development Mgr
Microsoft Corporation
donthom@microsoft.com

PDC03
Make the connection

Microsoft

Agenda

- Intro to SPOT
  - Video
- Technology factors
- CLR implementation details
  - Demo
- The future

How Does It Work?

content

msn direct

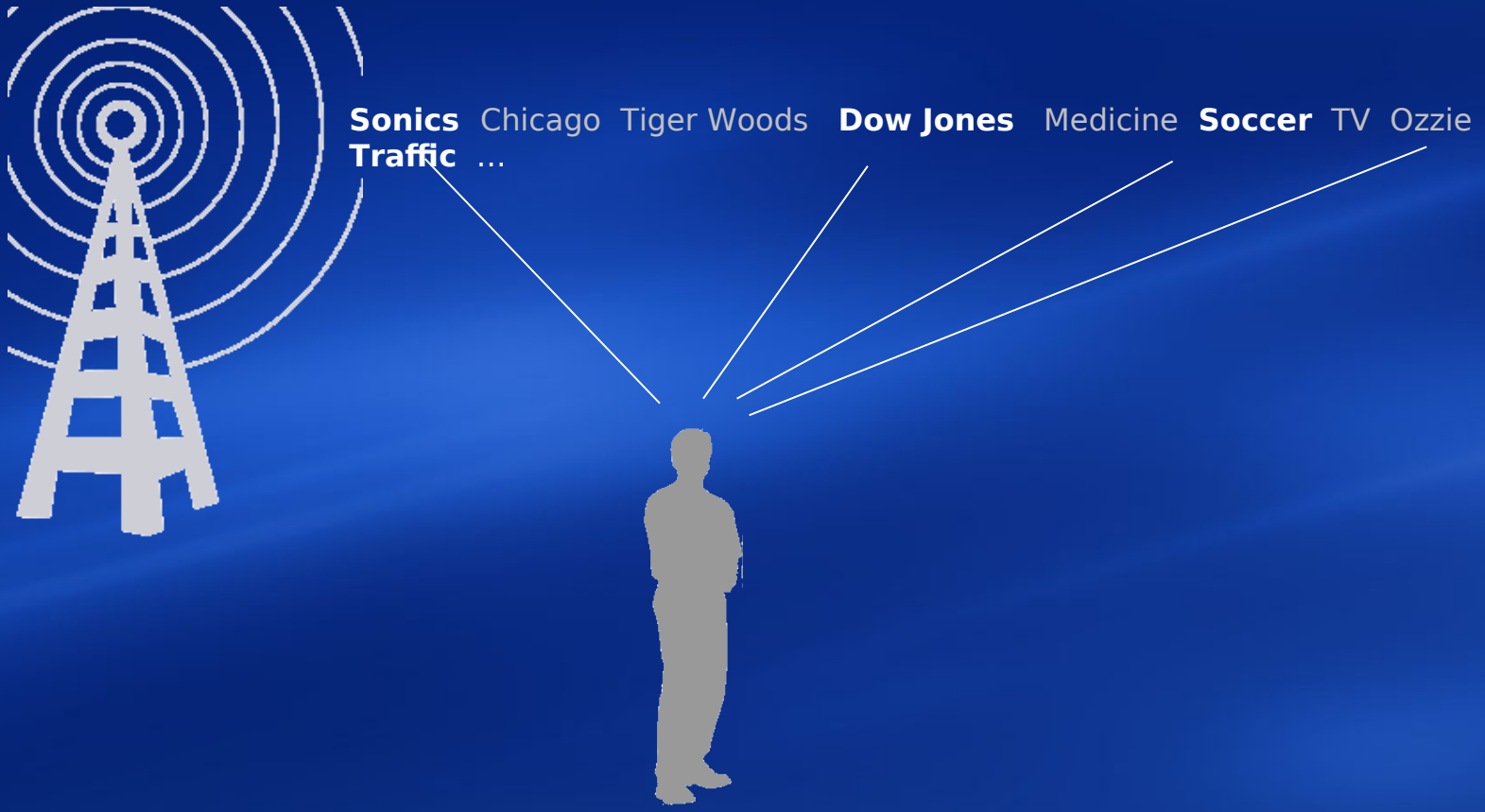
(network + service)

customer

Personalization

Sonics Chicago Tiger Woods Dow Jones Medicine Soccer TV Ozzie Traffic
...

Personalization



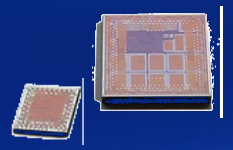
What John is interested automatically in…

MSN Direct

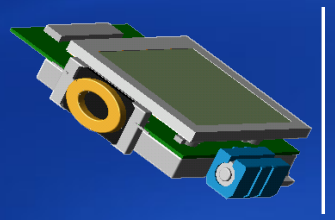
- Push vs. pull model
  - No need to seek out what you want
  - “Always On”
- Automatically delivered to your device
- Easy set up
  - Once personalized, you’re done
- Highly available
  - If you can get a radio signal, you can get your information
- Extensible
  - New services easily rolled-out

V1 Hardware Platform

- Custom silicon (2 chips)
  - “Stan”
    - Mixed analog / digital CMOS radio
    - Frequency agile
    - Burst receiver
  - “Ollie”
    - CPU: ARM7 TDMI-S
    - RAM: 384K



Stan Ollie



- Module
  - Busses
    - SPI, HPI, one wire, 32-bit parallel, USB, RS-232
  - Flash ROM (non-volatile): 1MB
  - EEPROM (optional)
  - LCD, battery (charge circuit, thermistor), buttons, piezo, vibrator, backlight, sensors, etc.

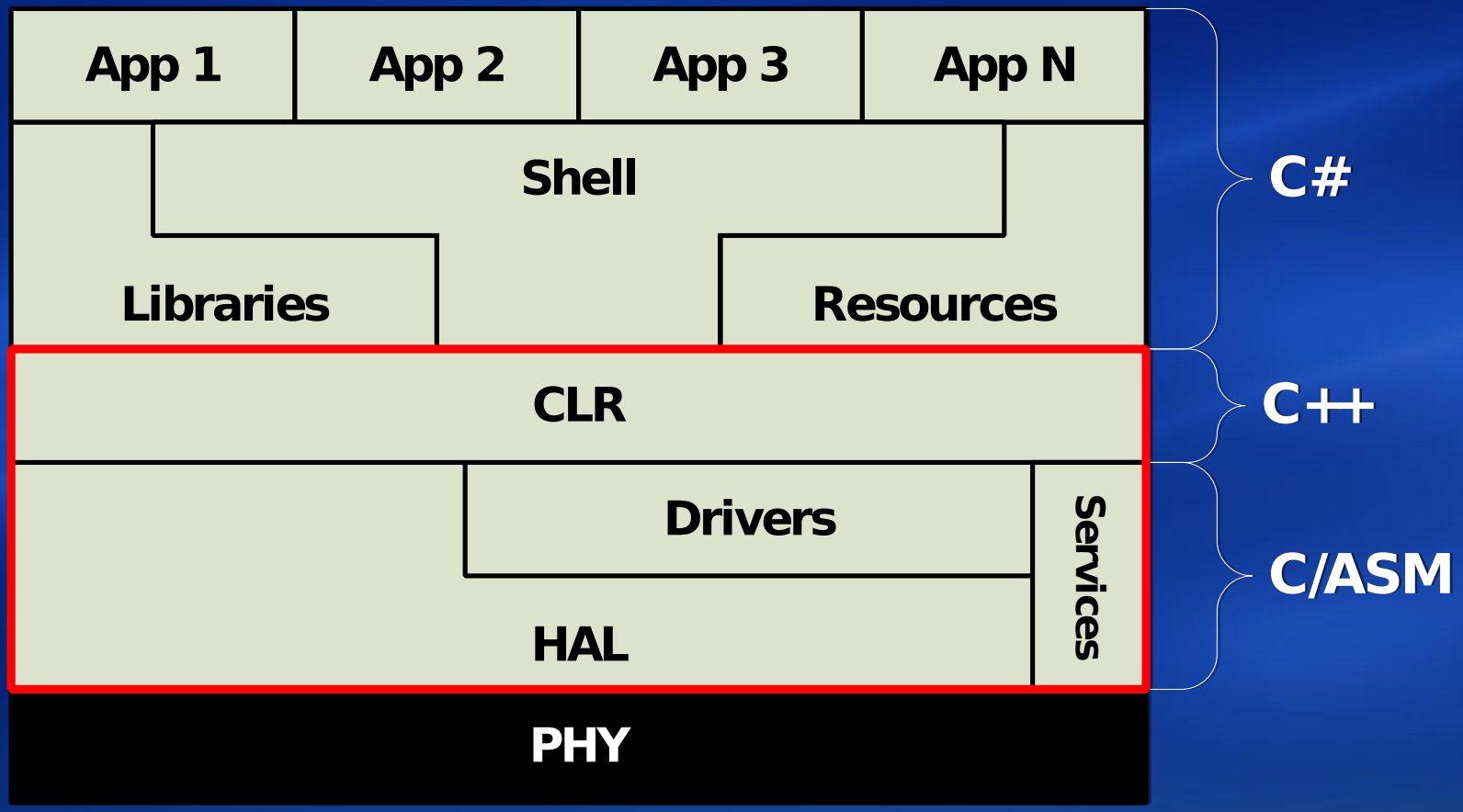
Technology Challenges

- Hardware
  - Limited energy
    - Target: 3 day runtime on 150mA-hr (@27Mhz)
  - Memory constraints (384K)
  - Arbitrary clock halting
    - Noise reduction during radio capture
  - Multi-model ROMs
    - Different drivers + HW

Technology Challenges

- Software
  - Safe programming model
  - Designed for OEM / ISV
    - 3rd party (embedded) development
  - Extensible platform
    - Chipsets, devices, sensors, applications
  - OTA updates
- Other
  - Time accuracy
    - 50ppm crystal accuracy, 5ppm time accuracy

Client Software Stack



HAL + CLR = OS

- CLR provides most functionality found in OS
  - Multi-threading/tasking
  - Memory management (garbage collection)
  - I/O
    - Needs HAL / drivers for physical access to peripherals
    - Benefits from native code efficiency
- HAL
  - Simple
    - No scheduler
    - No heaps
    - Minimal locking
  - Small
    - ~40K, including essential drivers
  - Efficient
    - Optimal method for given situation
  - Provides driver abstraction
    - Synchronous when energy efficiency dictates
    - Asynchronous for high latency operations

HAL / CLR Integration
Energy Efficiency

- Treat idle CPU as resource to consume
- Expecting low utilization == simpler code
  - Designing for low energy != design for max utilization
  - Cheap sleep
    - 4.7uSec to sleep CPU
  - DSR vs. “do it now” vs. sleep in place
    - Reducing clock speed while doing short

HAL / CLR Integration
Energy Efficiency (cont.)

- No polling
  - All hardware event-driven
- Hardware accelerators for radio processing
  - Parallel execution
  - Order of magnitude energy savings compared to native code (ARM)

HAL / CLR Integration
Execution Model

- Single application thread
  - No multi-tasking scheduler
  - Could make scheduler more sophisticated
    - No clear benefit for extra energy used to do so
- Interrupt Service Routines
  - Re-entrant
  - Mask-able
  - Prioritized
  - Never propagates into app
  - Worst case ISR latency ~200uSec

HAL / CLR Integration
Execution Model (cont.)

- Deferred Service Routines (DSR)
  - Continuations
    - Invoked when CLR is idle (sleep)
    - Round-robin scheduled with managed code execution
    - Drivers
      - Cooperative multi-tasking
      - Work is broken into <10ms chunks
      - State preserved for resumed execution
    - Callbacks
      - Use the continuation mechanism
      - Asynchronous notification
      - Registered when I/O requested
  - Completions
    - Timer ISR initiated (40uSec granularity)
    - Execute code at specific point in time
      - E.g., power regulator warm-up for burst radio capture

CLR Internals
Execution Model

- Managed code execution
  - Time sliced thread management
    - 20ms slices (preemptive)
    - Non-preemptive for unmanaged
  - Interpreted
    - Smaller footprint
    - Fine-grain control over execution
- Unmanaged code execution
  - Native methods interspersed with managed code
- Single implicit app domain
  - Support loading/unloading of assemblies

CLR Internals
Execution Model (cont.)

- Unmanaged code execution
  - Native methods interspersed with managed code
- Single implicit app domain
  - Support loading/unloading of assemblies

CLR Internals
Type System

- Only managed types and safe instructions
- Support of CLS and traditional .NET types
  - No support for:
    - Multi-dimensional (sparse, jagged) arrays
    - Machine-dependent types (element type I, U)
      - Allows unsafe conversions
  - Strings are represented in UTF-8
    - Exposed as Unicode
- Runtime treatment
  - Value types emulated by runtime (not 1st class)
    - Allows for GC optimization
    - Saves RAM

CLR Internals
Type System (cont.)

- Global, shared string table
  - Well-known values
    - Type names
    - Method names
    - Field names
  - Reduces xrefs between assemblies
- Patching
  - Method substitutions
  - Resource substitutions
  - New types
  - No type substitution
  - Tool
    - Diffs assemblies
    - Generates patch assembly

CLR Internals
Garbage Collection

- Balance between speed and size (RAM)
- Mark and sweep
  - Generational too heavy-weight
- Non-incremental
- Doesn’t require metadata
  - All data is self-describing
- Support for non-volatile storage

CLR Internals
SPOT Extensions

- “Not so weak” references
  - Prioritized
  - Time stamped
  - Persisted
    - RAM
      - Survives crash/reboots
      - CRC’d to protect against corruption
    - EEPROM or FlashROM
      - Survives power failures
  - Provides “automatic” data management
    - Old application “content” is GC’d like everything else
      - Priority is used by apps to help control order of disposal

CLR Internals
SPOT Extensions (cont.)

- “Weak” delegates
  - Runtime manages dangling refs
- Execution constraints (sub-threads)
- Serialization engine
  - Heavy reliance on attributes to guide data encoding
  - New attributes
    - Bit packed
    - Scaled
    - Array size
    - Not null
    - Fixed type

Developing For SPOT

demo

PDC03

Make the connection

CLR Internals
Future Work

- Increase framework support
- Reduce native code use
  - We can do a lot more in managed code than we do today
  - Careful analysis of tradeoffs with migrating functionality to managed code
    - Not a full re-architecture, but substantial rethinking of runtime structures/operation
  - More resources = more managed code
- Developer support
  - SDK, OAK, CDK, Stamp

Community

- Ask the Experts
  - Tuesday @ 7pm in Hall G, H
- Future of Mobility Panel
  - Thursday @ 1:45 in Hall G, H
- Send mail
  - donthom@microsoft.com

Microsoft®

© 2003-2004 Microsoft Corporation. All rights reserved.
This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.